Problema:

Data una sequenza di elementi in ordine qualsiasi, ordinarla.

- Questo è un problema fondamentale, che si presenta in moltissimi contesti, ed in diverse forme.
- Nel nostro caso formuliamo il problema in termini di ordinamento di vettori:

  Dato un vettore A di n elementi, ordinarlo in modo crescente
- Per semplicità faremo sempre riferimento a vettori di interi.

$$\Longrightarrow \begin{array}{cccccc} 5 & 2 & 4 & 6 & 1 & 3 \\ \\ 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

Ordinamento per inserzione (insertion sort)

- **Esempio:** Ordinamento di una mano di ramino
  - si inizia con la mano sinistra vuota e le carte coperte sul tavolo
  - si prende dalla tavola una carta alla volta e la si inserisce nella corretta posizione nella mano sinistra
  - ...
  - si termina quando si sono finite tutte le carte sul tavolo.
- Stesso procedimento per ordinare un vettore:
  - inizialmente il vettore rappresenta il mazzo sul tavolo
  - si usa un ciclo per analizzare uno alla volta gli elementi del vettore
  - Alla generica iterazione la situazione è la seguente

mano sinistra	carte ancora da scoprire

    ↑ nuova carta
  - per inserire la nuova carta al posto giusto nella mano sinistra dobbiamo
    - scorrere gli elementi che lo precedono per decidere la posizione che gli compete
    - spostare di un posto verso destra gli elementi maggiori per fargli spazio.

Esempio

In verde le carte ancora da esaminare, in rosso quelle già esaminate (mano sinistra). La nuova carta da esaminare è sottolineata.

<u>5</u>	2	4	6	1	3
5	<u>2</u>	4	6	1	3
2	5	<u>4</u>	6	1	3
2	4	5	<u>6</u>	1	3
2	4	5	6	<u>1</u>	3
1	2	4	5	6	<u>3</u>
1	2	3	4	5	6

- Una volta individuata la posizione k in cui inserire la nuova carta dobbiamo farle spazio, ovvero spostare verso destra di una posizione tutte le carte rosse da k in poi.

Esempio:

1	2	4	5	6	3
1	2	4	5	6	3
1	2	4	5		6
1	2	4		5	6
1	2		4	5	6

- A questo punto possiamo piazzare la carta in modo ordinato

1 2 3 4 5 6

- Definiamo allora una procedura che sposta tutti gli elementi di un vettore verso destra di una posizione tra due indici dati `from` e `to`

```
void shiftR(int v[], int from, int to)
{
int i;
for (i=to-1; i>=from; i--)
  v[i+1] = v[i];
}
```

- L'elemento in posizione `to` viene perso
- Bisogna procedere da destra verso sinistra (perché?)
- se `to` è minore o uguale a `from` non succede nulla

Possiamo allora definire la procedura di ordinamento per inserzione come segue

```
void sort(int v[], int dim)
{
int h, curr, j=1;
while (j<dim)
  { h=0;
    curr=v[j];
          /* curr e' l'elemento da piazzare */

    while((v[h]<curr) && (h<j))
       h++;
          /* curr va inserito in posizione h */

    shiftR(v,h,j);
     v[h]=curr;
    j++;
   }
}
```

Possiamo anche rendere la procedura più efficiente, nel modo seguente

```
void sort (int v[], int dim) {
  int i, j, prossimo;
  for (i = 1; i < dim; i++) {
     prossimo = v[i];
     j = i;
     while ((j > 0) && (v[j-1] > prossimo)) {
        v[j] = v [j-1];
        j--;
     }
     v[j] = prossimo;
  }
}
```

Ordinamento per selezione del minimo (selection sort)

- **Esempio:** Ordinamento di un mazzo di carte
  - si seleziona la carta più piccola e si mette da parte
  - delle rimanenti si seleziona la più piccola e si mette da parte
  - ...
  - si termina quando rimane una sola carta
- Ordinamento di un vettore:
  - per selezionare l'elemento più piccolo tra quelli rimanenti si utilizza un ciclo
  - mettere da parte significa scambiare con l'elemento che si trova nella posizione che compete a quello selezionato

- in verde la parte che rimane da analizzare
  in blu l'elemento minimo selezionato
  in marrone lo scambio effettuato
  in rosso la parte ordinata

5	2	4	6	1	3	
	5	2	4	6	1	3
	1	2	4	6	5	3
1	2	4	6	5	3	
	1	2	4	6	5	3
	1	2	4	6	5	3
1	2	4	6	5	3	
	1	2	4	6	5	3
	1	2	3	6	5	4
1	2	3	6	5	4	
	1	2	3	6	5	4
	1	2	3	4	5	6
1	2	3	4	5	6	
	1	2	3	4	5	6
	1	2	3	4	5	6
1	2	3	4	5	6	
1	2	3	4	5	6	

Implementazione

```
int minPos(int v[], int from, int to);
/* calcola la posizione del minimo elemento di
   v nella porzione [from,to]                */

void swap(int *p, int *q);
/* scambia le variabili puntate da p e q */

/** PROCEDURA DI ORDINAMENTO PER SELEZIONE **/

void sort(int v[], int dim)
{
  int i, min;
  for(i=0; i<dim-1; i++)
    {
      min = minPos(v, i, dim-1);
      swap(v+i, v+min);
    }
}
```

Scrivere per **esercizio** le procedure `swap` e `minpos`

```
int minPos(int v[], int from, int to) {
/* calcola la posizione del minimo elemento di
   v nella porzione [from,to]                */

    int i, pos;
    pos = from;
    for (i=from+1; i<=to; i++)
        if (v[i] < v[pos])
            pos = i;
    return pos;
}

void swap(int *p, int *q) {
/* scambia le variabili puntate da p e q */
    int temp = *p;
    *p = *q;
    *q = temp;
}
```

Ordinamento a bolle (bubble sort)

- Si fanno salire gli elementi più piccoli ("più leggeri") verso l'inizio del vettore ("verso l'alto"), scambiandoli con quelli adiacenti.
- L'ordinamento è suddiviso in n-1 fasi:
  - fase 0: 0° elemento (il più piccolo) in posizione 0
  - fase 1: 1° elemento in posizione 1
  - ...
  - fase n-2: $(n\text{-}2)^o$ elemento in posizione n-2, e quindi $(n\text{-}1)^o$ elemento in posizione n-1
- Nella fase i: cominciamo a confrontare dal basso e portiamo l'elemento più piccolo (più leggero) in posizione i

5	2	4	6	1	3	
	5	2	4	6	1	3
	5	2	4	1	6	3
	5	2	1	4	6	3
	5	1	2	4	6	3
	1	5	2	4	6	3
1	5	2	4	6	3	
	1	5	2	4	3	6
	1	5	2	3	4	6
	1	5	2	3	4	6
	1	2	5	3	4	6
1	2	5	3	4	6	
1	2	3	5	4	6	
1	2	3	4	5	6	
1	2	3	4	5	6	

```
/** PROCEDURA BUBBLE SORT **/

void sort(int v[], int dim)
{
  int temp,i,j;
  for (i = 0; i < dim-1; i++)     /* fase i-esima */

    for (j = dim-1; j > i; j--)   /* bolla piu' leggera in posizione i */
      if (v[j] < v[j-1])
        swap(v+j, v+j-1);
}
```

Ordinamenti ricorsivi
Selection Sort ricorsivo

- Il metodo del selection sort può essere facilmente realizzato in modo ricorsivo
- si definisce una procedura che ordina (ricorsivamente) la porzione di array individuata da due indici `from` e `to`
- il minimo elemento della porzione viene messo in posizione `from` per poi ordinare ricorsivamente la porzione tra `from+1` e `to`
- Il caso base corrisponde all'ordinamento di una porzione fatta da un solo elemento (è già ordinata)

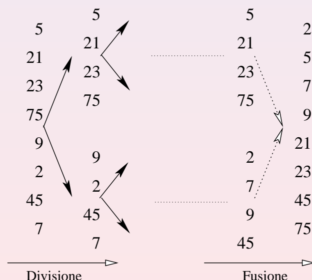
```
void SelectionSort(int v[], int from, int to){
  if (from < to) {
    int min = minPos(v,from,to);
    swap(v+from, v+min);
    SelectionSort(v, from+1, to);
  } }

void sort(int v[], int dim) {
    SelectionSort(v,0,dim-1);
}
```

Merge sort

Si divide il vettore da ordinare in due parti:

- si ordina ricorsivamente la prima parte
- si ordina ricorsivamente la seconda parte
- si combinano (operazione di fusione, merge) le due parti ordinate

Esempio:



Esprimiamo il procedimento in uno pseudo-linguaggio

```
ordina per fusione gli elementi di A da from a to

IF from < to (c'è più di un elemento tra from e to)

THEN
     mid = (from + to) /2
     ordina per fusione gli elementi di A da from a mid
     ordina per fusione gli elementi di A da mid +1 a to
     fondi
        gli elementi di A da from a mid con
        gli elementi di A da mid+1 a to
        restituendo il risultato nel sottovettore
         di A da from a to
```

Implementiamo l'algoritmo in `C`, definendo una procedura ricorsiva

```
void mergeRicorsivo(int A[], int from, int to)
```

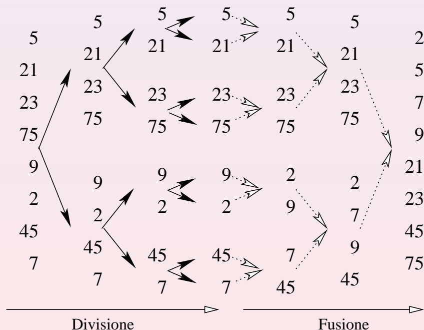
che ordina la porzione dell'array `A` individuata dagli indici `from` e `to`.

```
void mergeRicorsivo(int A[], int from, int to)

{
  int mid;

  if (from < to) {              /* l'intervallo da mid a to, estremi
                                   inclusi, comprende almeno due elementi */
    mid = (from + to) / 2;

    mergeRicorsivo(A, from, mid);
    mergeRicorsivo(A, mid+1, to);

    merge(A, from, mid, to);    /* fonde le due porzioni ordinate [from, mid],
                                   [mid+1, to] nel sottovettore [from, to]  */
  }
}
```

La procedura `mergeSort` che ordina un array di interi è semplicemente

```
void sort(int v[], int dim)
{
mergeRicorsivo(v, 0, dim-1);
}
```

Esempio:



- Vediamo l'operazione di *fusione*, definendo la procedura

  ```
  void merge(int A[], int from, int mid, int to)
  ```

  che fonde le due porzioni dell'array `A` con indici compresi tra `from` e `mid` e tra `mid+1` e `to`.
- La procedura utilizza un array di supporto `B`: per semplicità, supponiamo di avere una costante `LUNG` che definisce la lunghezza degli array che stiamo trattando.

```
void merge(int A[], int from, int mid, int to)
 {
  int B[LUNG];                                  /* vettore di appoggio */
  int primo, secondo, appoggio, da_copiare;

  primo = from;
  secondo = mid + 1;
  appoggio = from;

  while (primo <= mid && secondo <= to) { /* copia in modo ordinato      */
    if (A[primo] <= A[secondo]) {         /* gli elementi della prima e  */
      B[appoggio] = A[primo];             /* della seconda porzione in B */
      primo++;                            /* fino ad esaurire una delle due */
    }
    else {
      B[appoggio] = A[secondo];
      secondo++;
    }
    appoggio++;
  }
```

```
 if (secondo > to)             /* e‘ finita prima la seconda porzione */
 /* copia da A in B tutti gli elementi della
    prima porzione fino a mid */

    for (da_copiare = primo; da_copiare <= mid; da_copiare++) {
       B[appoggio] = A[da_copiare];
       appoggio++;
    }
  else                          /* e‘ finita prima la prima porzione */

    for (da_copiare = secondo; da_copiare <= to; da_copiare++) {
    /* copia da A in B tutti gli elementi della
    /* seconda porzione fino a to */

       B[appoggio] = A[da_copiare];
       appoggio++;
    }

/* ricopia tutti gli elementi da from a to da B ad A */
  for (da_copiare = from; da_copiare <= to; da_copiare++)
    A[da_copiare] = B[da_copiare];
}
```

Ricerca binaria in un vettore ordinato

Per cercare un elemento in un vettore ordinato, è possibile sfruttare l'ordinamento dimezzando ad ogni passo l'intervallo di elementi ancora da considerare per la ricerca.

Cerchiamo un elemento el in un vettore ordinato A:

- usiamo due indici from e to per delimitare l'intervallo di ricerca
- confrontiamo el con l'elemento di A al centro dell'intervallo, ovvero con A[mid], dove $mid = (from+to)/2$.
  Distinguiamo 3 casi:
  - $el = A[mid] \Longrightarrow$ l'elemento è stato trovato
  - $el < A[mid] \Longrightarrow$ cerchiamo nell'intervallo con indici da from a $mid-1$
  - $el > A[mid] \Longrightarrow$ cerchiamo nell'intervallo con indici da $mid+1$ a to
- se non troviamo el ci fermiamo quando l'intervallo di ricerca si è annullato (ovvero, $from > to$)

Esempio: Ricerca di 24 nel seguente vettore ordinato.

14	19	24	31	33	40	45	52	60
from								*to*
				mid				
from			*to*					
	mid							
		from	*to*					
		mid						

```
ricerca el nella parte di A compresa tra from e to
if to < from
then l'elemento non compare in A
else mid = (from+to)/2 (divisione intera)
if el = A[mid]
then l'elemento è stato trovato
el < A[mid]
then ricerca el nella parte di A tra from e mid-1
else ricerca el nella parte di A tra mid+1 e to
```

```
int RicercaBinariaRic(int A[], , int el, int from, int to)

/* Funzione ricorsiva che restituisce la posizione di el nella parte di A
    compresa fra from e to; restituisce -1 se el non sta nella porzione*/

{ int mid, posizione;

  if (from > to)
   posizione = -1;          /* la parte del vettore tra from e to e‘ vuota */
 else {
   mid = (from + to) / 2;
   if (el == A[mid]) {                    /* l’elemento e‘ stato trovato */
     posizione = mid;
   }
   else
     if (el < A[mid])                     /* cerca nella parte inferiore */
       posizione = RicercaBinariaRic(A, el, from, mid-1);
     else                                 /* cerca nella parte superiore */
       posizione = RicercaBinariaRic(A, el, mid+1, to);
 }
 return posizione;
}
```

```
int RicercaBinaria(int A[], int el, int n)
 /* Effettua la ricerca binaria di el nei primi n elementi di A, usando la
    funzione ricorsiva RicercaBinariaRic.  Restituisce la posizione di el se
    questo e‘ presente in A, -1 altrimenti.  */
{
 return RicercaBinariaRic(A, el, 0, n-1);
 }
```